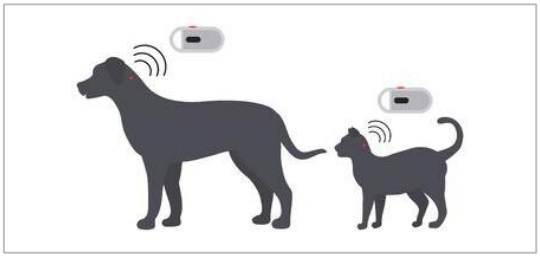
**Universidad Tecnológica Nacional**

**Tecnicatura Universitaria en Programación a Distancia**

**Programación II**

# TRABAJO FINAL INTEGRADOR



## Tema: MASCOTA – MICROCHIP


## Integrantes y Roles:

- **Arjona, Paola Yanina** *Analista de Dominio y Responsable de Entidades*
- **Arjona, Martin Joel** *Encargado de Acceso a Datos (DAO) y Persistencia*
- **Paco L, Rosa Lourdes** *Coordinador de reglas de negocio y Validaciones*
- **Reynoso, Thomás** *Interfaz y Flujo de la Aplicación - Conector del sistema*

*Si bien cada integrante tuvo un rol definido, el proyecto se desarrolló con aportes conjuntos para asegurar coherencia y correcto funcionamiento.*

**Profesor:** Alberto Cortez

**Tutor:** Juan Cruz Robledo

**Comisión:** 17

**Grupo**: 111


17/11/2025

# ÍNDICE

1. Introducción ..... 1

2. Objetivo ..... 1

3. Elección del dominio ..... 1

4. Diseño y UML ..... 2

5. Persistencia y Base de Datos (MySQL) ..... 2

6. Arquitectura del proyecto ..... 3

7. Validaciones y reglas de negocio ..... 5

8. Pruebas realizadas ..... 6

9. Conclusión ..... 8

10. Mejoras futuras ..... 8

# 1. Introducción

El presente trabajo tiene como objetivo desarrollar una aplicación en Java que implemente una **arquitectura por capas**, con persistencia de datos mediante **JDBC** y una relación **unidireccional 1→1** entre las entidades **Mascota** y **Microchip**. Se busca integrar conceptos de modularidad, reutilización y buenas prácticas en la programación orientada a objetos, utilizando DAOs genéricos y concretos, servicios para la lógica de negocio, y pruebas unitarias para verificar la funcionalidad del sistema.

El proyecto permite crear, consultar, actualizar y eliminar registros de mascotas y microchips, garantizando la integridad de la relación entre ambas entidades y mostrando la relevancia de un diseño escalable y mantenible.

# 2. Objetivo

Desarrollar una aplicación Java estructurada con una arquitectura por capas que integre persistencia mediante JDBC y aplique correctamente una relación unidireccional 1→1 entre entidades del dominio.

Para ello, se busca implementar una organización modular que separe la lógica de negocio, el acceso a datos y la presentación, garantizando mantenibilidad y escalabilidad.

También se propone diseñar un esquema de base de datos en MySQL que respete integridad referencial y permita gestionar transacciones de forma segura, incluyendo operaciones atómicas para la creación, modificación y eliminación lógica de registros. Finalmente, se pretende demostrar el funcionamiento completo del sistema a través de un flujo básico de operaciones sobre Mascotas y Microchips, validando la comunicación entre capas y el correcto manejo de conexiones, excepciones y reglas de negocio.

# 3. Elección del dominio

**Justificación:** Se seleccionó el dominio Mascota–Microchip por ser un caso real y representativo de sistemas de registro animal. Es adecuado para demostrar la relación unidireccional 1→1:

- Cada mascota puede tener un único microchip.
- Cada microchip pertenece a una única mascota.

Este dominio incluye datos obligatorios, fechas y claves únicas, lo que permite aplicar reglas de negocio y validaciones. El sistema permite registrar mascotas y microchips, asociarlos en forma estricta 1→1, actualizar información, realizar bajas lógicas, listar registros activos, buscar mascotas por nombre, microchips por ID, manejar errores y entradas inválidas.

## 4. Diseño y UML

La relación **Mascota** → **Microchip** se refleja con un atributo privado microchip en Mascota (unidireccional).
Decisión de diseño 1→1: se implementa **clave foránea única en Microchip** (a_id con UNIQUE y FK a Mascota(id)) y ON DELETE CASCADE. Esto asegura que cada Mascota tiene como máximo un Microchip y mantiene la integridad referencial en la base. Ver anexo UML.

## 5. Persistencia y Base de Datos (MySQL)

La base de datos se implementó en MySQL con conjunto de caracteres UTF8MB4, asegurando soporte completo para la relación unidireccional 1→1 entre Mascota y Microchip. Cada tabla posee su clave primaria (id_mascota y id_microchip), campos obligatorios, atributos opcionales y el campo eliminado para bajas lógicas. La relación 1→1 se implementa mediante microchip_id en la tabla mascota con restricción UNIQUE y clave foránea FOREIGN KEY ... ON DELETE CASCADE ON UPDATE CASCADE, asegurando que un microchip solo pertenezca a una mascota y que la eliminación se propague correctamente.

El **control de transacciones** se realiza en la capa de servicio (MascotaService y MicrochipService). Cada operación que modifica datos (insertar, actualizar o eliminar) abre una transacción sobre la conexión JDBC:

- Se ejecutan las operaciones del DAO correspondientes.
- Si todas finalizan correctamente, se realiza commit().
- Si ocurre algún error, se ejecuta rollback(), revirtiendo todos los cambios de la operación para mantener la integridad de la base de datos.

En **operaciones compuestas**, como crear una mascota junto con su microchip, ambas inserciones comparten la misma conexión y transacción. Esto garantiza que si falla cualquiera de las dos operaciones, **ninguna de las entidades queda parcialmente registrada**, cumpliendo la regla de negocio 1→1 y asegurando consistencia entre tablas.

Los scripts de prueba incluyen: microchips completos, mascotas con y sin microchip, microchips disponibles, y consultas de verificación (LEFT JOIN) para detectar violaciones de la relación 1→1 y contar registros activos.

# 6. Arquitectura del proyecto

```
TPI_Programacion_2
 src/
 ├── config/
 │   └── DatabaseConnection.java    # Gestión de conexiones a BD
 ├── dao/
 │   ├── GenericDao.java            # Interfaz genérica
 │   ├── MascotaDAO.java            # DAO concreto para Mascota (A)
 │   └── MicrochipDAO.java          # DAO concreto para Microchip (B)
 ├── entities/
 │   ├── Mascota.java               # Entidad A (relación 1→1)
 │   └── Microchip.java             # Entidad B
 ├── service/
 │   ├── GenericService.java        # Interfaz base con operaciones CRUD comunes
 │   ├── MascotaService.java        # Lógica de negocio para Mascota
 │   └── MicrochipService.java      # Lógica de negocio para Microchip
 ├── test/
 │   └── TestGenericDAO.java                    # Pruebas de la implementación
 │   └── TestTransaccionesObligatorias.java     # Pruebas de transacciones obligatorias
 └── tpi_programacion_2/
   └── AppMenu.java                 # Consola
   └── TPI_Programacion_2.java      # Aplicación principal
```

**1. Entities (entities/)**

**-Mascota.java**: representa la entidad Mascota con atributos id, nombre, especie, raza, dueño, fechaNacimiento.

-**Microchip.java**: representa la entidad Microchip con atributos id, codigo, fechaImplantacion, referenciando una Mascota (1→1).

**2. DAO (dao/)**

**-GenericDao.java**: es una interfaz genérica que define las operaciones CRUD básicas (crear, leer, actualizar y eliminar) para cualquier entidad del proyecto, incluyendo versiones que aceptan una **conexión externa** para permitir transacciones coordinadas (commit/rollback) en MySQL. La eliminación se implementa como **baja lógica** y todos los métodos lanzan SQLException para un manejo robusto de errores. Esta interfaz asegura consistencia entre los DAOs concretos (MascotaDAO, MicrochipDAO) y facilita la implementación de la **relación 1→1 unidireccional**, ya que permite que varias operaciones sobre entidades relacionadas compartan la misma transacción y mantengan la integridad referencial.

-**MascotaDAO y MicrochipDAO** son las implementaciones concretas de GenericDao para las entidades **Mascota** y **Microchip**. Cada DAO utiliza **PreparedStatement** para ejecutar consultas SQL sobre MySQL. Ambos soportan operaciones con **conexión externa**, permitiendo que la creación, actualización o eliminación de una Mascota y su Microchip asociado se realice dentro de la **misma transacción** (commit/rollback). Los DAOs

gestionan la persistencia de la relación **1→1 unidireccional** (Mascota → Microchip), y aplican **baja lógica** al eliminar registros.

**3. Service (service/)**

**-Interfaz** GenericService<T>
Define el comportamiento básico que deben implementar todos los servicios del proyecto, estandarizando las operaciones de negocio sobre cualquier entidad. Incluye métodos para insertar, actualizar y eliminar elementos dentro de transacciones con control automático de *commit* y *rollback*, asegurando integridad al trabajar con MySQL mediante JDBC. Provee operaciones de lectura (getById y getAll) que no requieren transacción por tratarse de consultas. Esta interfaz permite desacoplar la lógica de negocio del acceso a datos.

-**MascotaService** y **MicrochipService (Servicios de Negocio)**
A diferencia de GenericService, que define únicamente las operaciones CRUD obligatorias y la estructura general de manejo transaccional, los servicios concretos (MascotaService y MicrochipService) **implementan lógica de negocio real**, específica para cada entidad.

Ambos servicios aplican validaciones del dominio (unicidad del microchip, datos obligatorios de la mascota, coherencia de fechas), coordinan transacciones MySQL y delegan al DAO solo la persistencia, manteniendo un claro desacoplamiento. MascotaService además gestiona operaciones combinadas para la relación 1→1, mientras que MicrochipService se centra en asegurar la integridad y formato del identificador. Así, la capa de servicio agrega reglas y comportamientos que no pueden resolverse solo con CRUD.

**4. Configuración (config/)**

**-DatabaseConnection.java**: gestiona la conexión JDBC a la base de datos MySQL, cargando el driver y los parámetros de conexión.

**5. Test (test/)**

**-TestTransaccionesObligatorias:**

Demuestra el funcionamiento de las **transacciones y validaciones de la capa Service**. Verifica que MascotaService y MicrochipService implementen GenericService, que las operaciones CRUD y compuestas usen **commit/rollback**, y que se cumplan las **validaciones de campos obligatorios, formatos y la regla 1→1** (un Microchip no puede asociarse a más de una Mascota). Permite evidenciar la integridad de la relación unidireccional 1→1 y la correcta gestión de MySQL.

**-TestGenericDAO**

Enfocada en la **implementación de GenericDao y las operaciones CRUD** sobre Mascota y Microchip. Comprueba que los DAOs concretos cumplan la interfaz, que las operaciones individuales y compuestas funcionen con transacciones, y que se mantenga la **regla 1→1**. Sirve para validar la persistencia, integridad de datos y manejo de transacciones en MySQL de forma automática.

**6. Aplicación principal (tpi_programacion_2/TPI_Programacion_2.java)**

**-TPI_Programacion_2** funciona como punto de entrada de la aplicación. Su responsabilidad principal es iniciar el sistema, mostrar la presentación inicial y verificar que la conexión a la base de datos esté disponible antes de ejecutar cualquier operación.

Una vez validada la conexión, crea una instancia de **AppMenu** y delegando toda la lógica de interacción con el usuario. También gestiona posibles errores críticos de conexión y garantiza el cierre seguro de los recursos de base de datos al finalizar el programa.

**-AppMenu** implementa el menú principal de la aplicación y gestiona toda la interacción del usuario mediante consola. Ofrece un conjunto completo de operaciones CRUD tanto para *Mascotas* como para *Microchips*, además de búsquedas específicas y una operación especial que permite crear una mascota junto con su microchip en una misma transacción.

Este menú se encarga de mostrar todas las opciones disponibles organizadas por secciones ,leer y validar la entrada del usuario, ejecutar las acciones solicitadas invocando a MascotaService y MicrochipService y manejar errores de forma controlada para evitar interrupciones en la aplicación.

## 7. Validaciones y reglas de negocio

Las validaciones y reglas de negocio se implementan principalmente en la capa de servicio (MascotaService y MicrochipService), garantizando que los datos cumplan los requisitos del dominio antes de persistirlos en la base de datos. Principales validaciones:

1. **Campos obligatorios y longitudes máximas:**
   En Mascota: nombre, especie, dueño son obligatorios; raza y fechaNacimiento opcionales.
   En Microchip: codigo es obligatorio; veterinaria y observaciones opcionales. Cada campo verifica longitud máxima para evitar errores de truncado en MySQL.

2. **Unicidad del microchip (1→1)**
   Antes de insertar o actualizar un microchip, se consulta si ya existe en la base de datos otro registro con el mismo código.
   Se evita asociar un microchip a más de una mascota, cumpliendo la regla de negocio unidireccional 1→1.

3. **Coordinación de transacciones en operaciones compuestas**
   Operaciones que involucran varias entidades, como crear una mascota junto con su microchip, se realizan en la misma transacción.
   En caso de error en cualquiera de las operaciones, se ejecuta rollback(), asegurando que **ningún registro quede incompleto** y manteniendo la integridad referencial. Si todas las operaciones son correctas, se realiza commit(), guardando los cambios en la base de datos de manera atómica.

4. **Bajas lógicas**
   Tanto Mascota como Microchip poseen un campo eliminado. Las operaciones de eliminación no borran físicamente el registro, sino que lo marcan como inactivo (eliminado = true), conservando el historial y evitando inconsistencias.

Estas validaciones, junto con los DAOs que ejecutan las consultas SQL, aseguran que los datos cumplan las restricciones del dominio y que la relación 1→1 permanezca consistente en todo momento.

## 8. Pruebas realizadas

**Menú del programa JAVA**

a) Lista de opciones a seleccionar en el sistema (1 al 16 y 0 para salir)

```
=== SISTEMA DE GESTION DE MASCOTAS ===
Seleccione una opcion:

=== GESTION DE MASCOTAS ===
1.  Crear mascota
2.  Leer mascota por ID
3.  Listar todas las mascotas
4.  Actualizar mascota
5.  Eliminar mascota (logico)
6.  Buscar mascotas por nombre
7.  Buscar mascotas por especie
```

```
=== GESTION DE MICROCHIPS ===
8.  Crear microchip
9.  Leer microchip por ID
10. Listar todos los microchips
11. Actualizar microchip
12. Eliminar microchip (logico)
13. Buscar microchip por codigo
14. Buscar microchips por veterinaria
```

```
=== OPERACIONES ESPECIALES ===
15. Crear mascota CON microchip (transaccion)
16. Estadisticas del sistema

0.  Salir
```

**Ejemplo: Crear mascota**

a) Elección por parte del usuario : 1

```
Ingrese su opcion: 1

=== CREAR NUEVA MASCOTA ===
Nombre: Lola
Especie: Perro
Raza [opcional]: Labrador
Due?o: Lou
Fecha de nacimiento (AAAA-MM-DD) [opcional]: 2020-08-31
```

b) Confirmación y retorno al sistema

```
? EXITO: Mascota creada correctamente.
   ID generado: 23
   Nombre: Lola

=== SISTEMA DE GESTION DE MASCOTAS ===
Seleccione una opcion:
```

c) Registro en Base de Datos **- Tabla Mascotas-MySQL**

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| id_mascota | eliminado | nombre | especie | raza | fecha_nacimiento | duenio | microchip_id |
|---|---|---|---|---|---|---|---|
| 20 | 0 | sasi | Gato | Callejero | 2020-10-01 | Ivan | 16 |
| 21 | 0 | Lisa | Perro | Labrador | 2025-01-01 | Ivan | 17 |
| 22 | 0 | Dingo | Perro | Labrador | 2024-10-10 | Camino pepe | NULL |
| 23 | 0 | Lola | Perro | Labrador | 2020-08-31 | Lou | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

**Ejemplo Crear Microchip con código existente y asignado**

a) Elección por parte del usuario : 8

```
Ingrese su opcion: 8

=== CREAR NUEVO MICROCHIP ===
Codigo del microchip: MC006-ARG-2024
Fecha de implantacion (AAAA-MM-DD) [opcional]: 2025-11-15
Veterinaria [opcional]: Veterinaria Mundo Animal
Observaciones [opcional]: Evidencia de transaccion
Conexion a MySQL establecida exitosamente
Base de datos: MySQL
```

b) Se detecta el código existente, no se guarda el registro y retorna al sistema con enter

```
? ERROR AL CREAR MICROCHIP
? DETALLE: Error al insertar microchip: Ya existe un microchip con el c?digo: MC006-ARG-2024

? Presione Enter para continuar...
```

**Ejemplo Actualizar Microchip por otro ya asignado**

a) Elección por parte del usuario : 11, ID=10

```
Ingrese su opcion: 11

=== ACTUALIZAR MICROCHIP ===
Ingrese el ID del microchip a actualizar: 10
Conexion a MySQL establecida exitosamente
Base de datos: MySQL
```

b) salida de la Información del ID=10

```
Microchip actual:
ID: 10
Codigo: MC010-ARG-2024
Fecha Implantaci?n: 2024-03-15
Veterinaria: Centro de Adopci?n
Observaciones: Reservado para gato en recuperaci?n
```

c) Detección de código de microchip ya existente, no actualiza la modificación y vuelve al sistema.

```
Ingrese los nuevos datos (Enter para mantener el valor actual):
Codigo [MC010-ARG-2024]: MC013-ARG-2024
Fecha implantacion [2024-03-15] (AAAA-MM-DD): 2024-03-15
Veterinaria [Centro de Adopci▯n]: Centro de Adopcion
Observaciones [Reservado para gato en recuperaci▯n]: prueba rollback
Conexion a MySQL establecida exitosamente
Base de datos: MySQL

? ERROR AL ACTUALIZAR MICROCHIP
? DETALLE: Error al actualizar microchip: Ya existe otro microchip con el c▯digo: MC013-ARG-2024

? Presione Enter para continuar...
```

## 9. Conclusión

El proyecto permitió aplicar eficazmente los principios de arquitectura en capas, logrando una separación clara entre entidades, DAOs, servicios y la capa de presentación. La incorporación de JDBC y MySQL demostró la importancia del manejo riguroso de conexiones, transacciones y errores para asegurar un sistema confiable. La relación 1→1 entre Mascota y Microchip se implementó de manera consistente, combinando restricciones de base de datos con validaciones en la capa de servicio.

En conjunto, el desarrollo evidenció cómo una estructura modular facilita la comprensión del código, simplifica el mantenimiento y permite extender funcionalidades sin comprometer el diseño general del sistema. Además, se verificó que la integración entre las capas funciona correctamente y que el manejo de operaciones críticas —como la inserción conjunta de Mascota y Microchip— se resolvió de forma segura mediante commit y rollback

## 10. Mejoras futuras

El sistema cumple con los objetivos planteados, pero puede seguir mejorando. Algunas posibles ampliaciones son:

- Interfaz más amigable: reemplazar el menú por consola por una interfaz gráfica (JavaFX) o una versión web para mejorar la experiencia del usuario.
- Consultas y reportes avanzados: incorporar filtros, estadísticas simples y reportes específicos (por especie, estado del microchip, fechas, etc.).
- Escalabilidad: implementar paginación y optimizaciones en base de datos para mejorar el rendimiento ante grandes volúmenes de registros.

# Bibliografía


- Oracle. (2024). Java SE Documentation. Oracle Corporation.
- Oracle. (2024). JDBC API Guide. Oracle Corporation.
- MySQL. (2024). MySQL 8.0 Reference Manual. Oracle Corporation.
- Deitel, P. J., & Deitel, H. M. (2016). Cómo programar en Java (10.ª ed.). Pearson Educación.
- Fowler, M. (2003). Patterns of enterprise application architecture. Addison-Wesley.


Herramientas utilizadas

- Java SE 8+ – Lenguaje principal del proyecto.
- NetBeans IDE – Entorno de desarrollo utilizado para organizar el proyecto por capas y ejecutar la aplicación.
- MySQL Server 8.0 – Sistema gestor de base de datos relacional.
- MySQL Workbench
- MySQL Connector/J (Driver JDBC)

# ANEXO

## Diagrama UML